CS 267
Dense Linear Algebra:
Parallel Gaussian Elimination

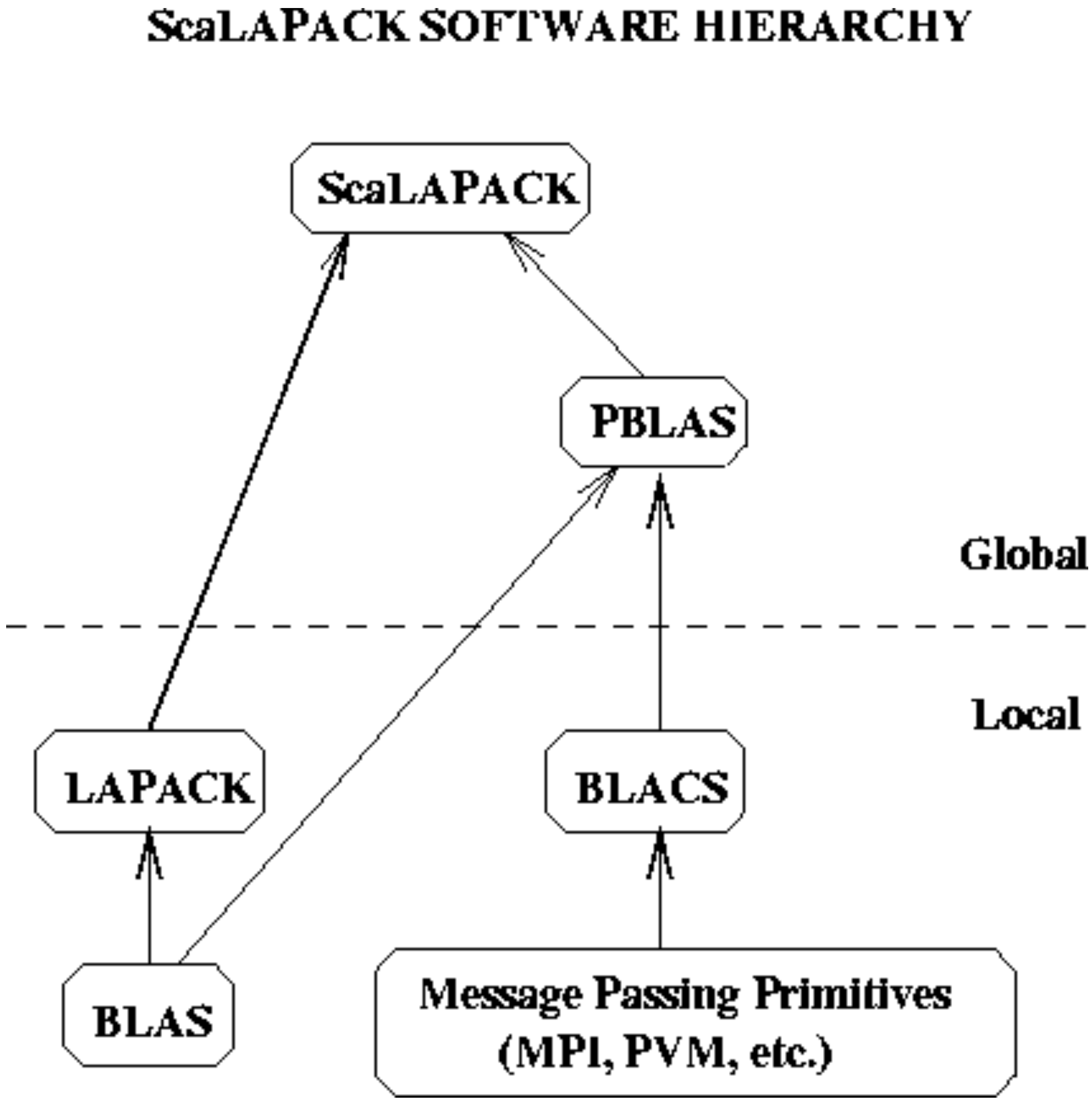
James Demmel

www.cs.berkeley.edu/~demmel/cs267_Spr08

Outline

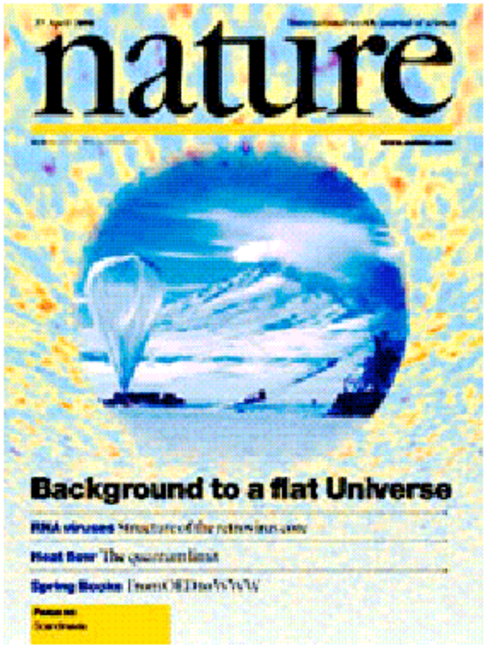
- Motivation, overview for Dense Linear Algebra
- Review Gaussian Elimination (GE) for solving Ax=b
- Optimizing GE for caches on sequential machines
    - using matrix-matrix multiplication (BLAS)
- LAPACK library overview and performance
- Data layouts on parallel machines
- Parallel Gaussian Elimination
- ScaLAPACK library overview
- Eigenvalue problems
- Current Research

Sca/LAPACK Overview



Success Stories for Sca/LAPACK

- **Widely used**
  - **Adopted by Mathworks, Cray, Fujitsu, HP, IBM, IMSL, NAG, NEC, SGI, …**
  - **>84M(56M in 2006) web hits @ Netlib (incl. CLAPACK, LAPACK95)**
- **New Science discovered through the solution of dense matrix systems**
  - **Nature article on the flat universe used ScaLAPACK**
  - **Other articles in Physics Review B that also use it**
  - **1998 Gordon Bell Prize**
  - **www.nersc.gov/news/reports/newNERSCresults050703.pdf**



Cosmic Microwave Background Analysis, BOOMERanG collaboration, MADCAP code (Apr. 27, 2000).

ScaLAPACK

Motivation (1)

3 Basic Linear Algebra Problems

1. Linear Equations: Solve $Ax=b$ for x
2. Least Squares: Find x that minimizes $\|r\|_2 \equiv \sqrt{\Sigma\, r_i^2}$ where $r=Ax-b$
   - Statistics: Fitting data with simple functions

3a. Eigenvalues: Find λ and x where $Ax = \lambda\, x$

- Vibration analysis, e.g., earthquakes, circuits

3b. Singular Value Decomposition: $A^T Ax=\sigma^2 x$

- Data fitting, Information retrieval

Lots of variations depending on structure of A

- A symmetric, positive definite, banded, …

Motivation (2)

- **Why dense A, as opposed to sparse A?**
  - **Many large matrices are sparse, but …**
  - **Dense algorithms easier to understand**
  - **Some applications yields large dense matrices**
  - **LINPACK Benchmark (www.top500.org)**
    - **"How fast is your computer?" = "How fast can you solve dense Ax=b?"**
  - **Large sparse matrix algorithms often yield smaller (but still large) dense problems**

Current Records for Solving Dense Systems (2007)

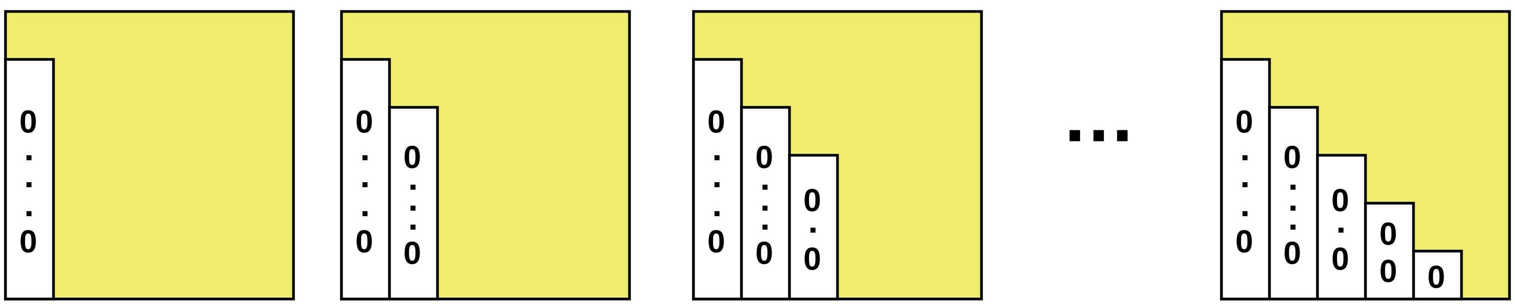
www.netlib.org, click on Performance Database Server

Machine	Gigaflops n=100	n=1000	Any n	Peak
IBM BlueGene/L (213K procs)			478K (478 Teraflops) (n=2.5M)	596K
NEC SX 8				
(8 proc, 2 GHz)		75.1		128
(1 proc, 2 GHz)	2.2	15.0		16
…				
Palm Pilot III	.00000169 (1.69 Kiloflops)			

Gaussian Elimination (GE) for solving Ax=b

- **Add multiples of each row to later rows to make A upper triangular**
- **Solve resulting triangular system Ux = c by substitution**

```
… for each column i
… zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
   … for each row j below row i
   for j = i+1 to n
       … add a multiple of row i to row j
       tmp = A(j,i);
       for k = i to n
            A(j,k) = A(j,k) - (tmp/A(i,i)) * A(i,k)
```



After i=1　　After i=2　　After i=3　　…　　After i=n-1

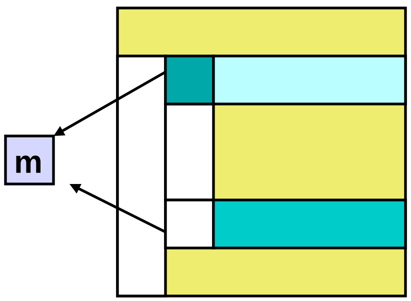
Refine GE Algorithm (1)

- **Initial Version**

```
… for each column i
… zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
    … for each row j below row i
    for j = i+1 to n
        … add a multiple of row i to row j
        tmp = A(j,i);
        for k = i to n
            A(j,k) = A(j,k) - (tmp/A(i,i)) * A(i,k)
```

- **Remove computation of constant tmp/A(i,i) from inner loop.**

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i to n
            A(j,k) = A(j,k) - m * A(i,k)
```



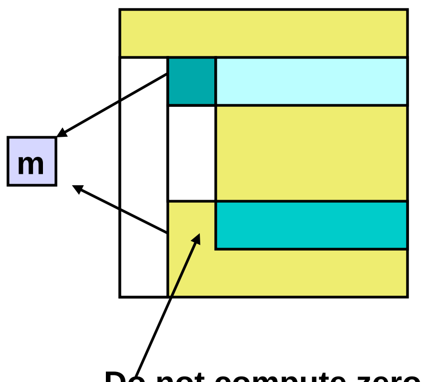
Refine GE Algorithm (2)

- **Last version**

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i to n
            A(j,k) = A(j,k) - m * A(i,k)
```

- **Don't compute what we already know: zeros below diagonal in column i**

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - m * A(i,k)
```



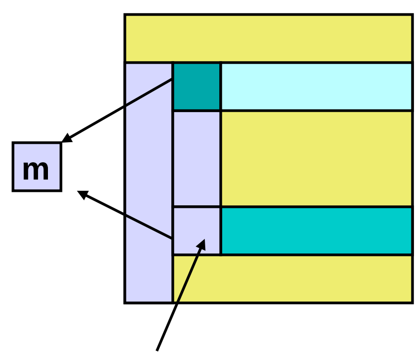
Refine GE Algorithm (3)

- Last version

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - m * A(i,k)
```

- Store multipliers m below diagonal in zeroed entries for later use

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```



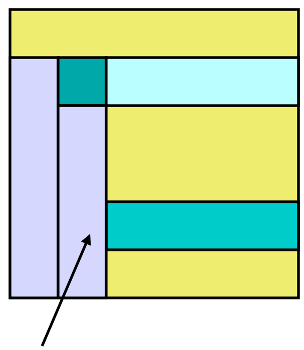
Refine GE Algorithm (4)

- **Last version**

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

- **Split Loop**

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
    for j = i+1 to n
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

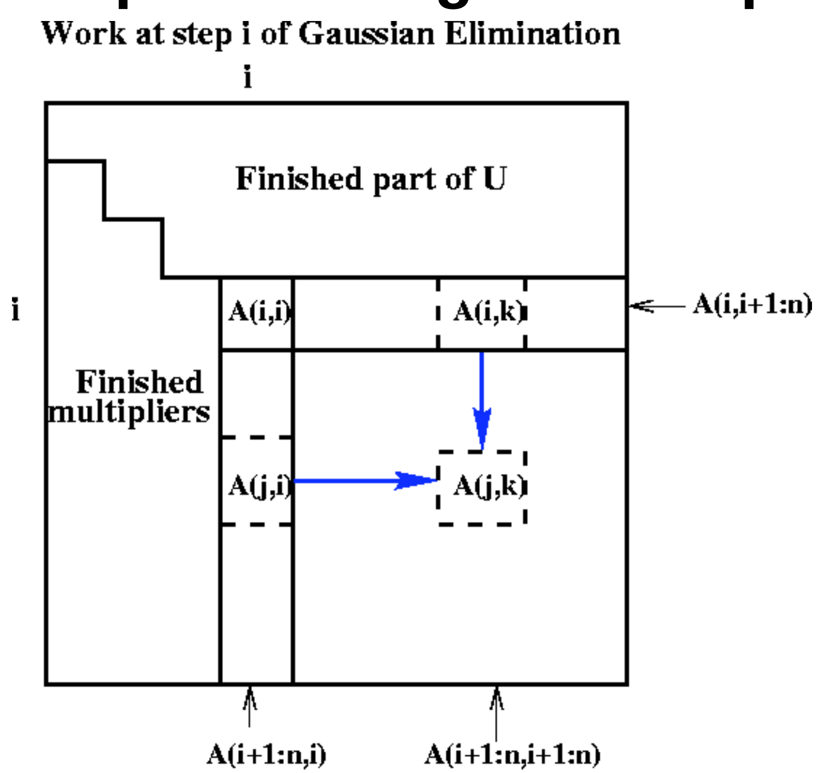


Refine GE Algorithm (5)

- **Last version**

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
    for j = i+1 to n
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

- **Express using matrix operations (BLAS)**




```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) * ( 1 / A(i,i) )
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )
         - A(i+1:n , i) * A(i , i+1:n)
```

What GE really computes

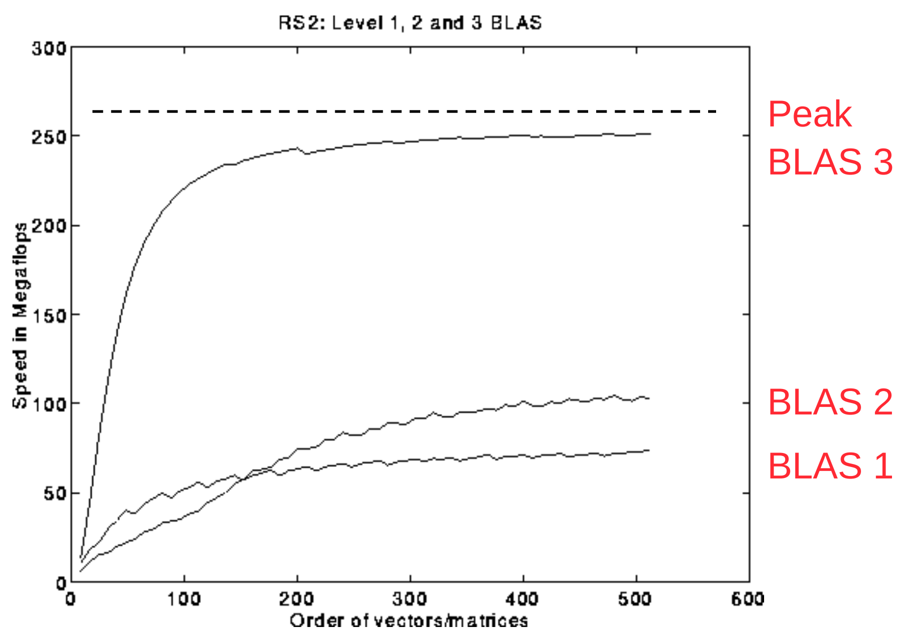
```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n ) - A(i+1:n , i) * A(i , i+1:n)
```

- Call the strictly lower triangular matrix of multipliers M, and let L = I+M
- Call the upper triangle of the final matrix U
- *Lemma (LU Factorization):* If the above algorithm terminates (does not divide by zero) then A = L*U
- Solving A*x=b using GE
  - Factorize A = L*U using GE (cost = 2/3 n^3 flops)
  - Solve L*y = b for y, using substitution (cost = n^2 flops)
  - Solve U*x = y for x, using substitution (cost = n^2 flops)
- Thus A*x = (L*U)*x = L*(U*x) = L*y = b as desired

Problems with basic GE algorithm

- **What if some A(i,i) is zero? Or very small?**
  - **Result may not exist, or be "unstable", so need to pivot**
- **Current computation all BLAS 1 or BLAS 2, but we know that BLAS 3 (matrix multiply) is fastest (earlier lectures…)**

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)          … BLAS 1 (scale a vector)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )  … BLAS 2 (rank-1 update)
            - A(i+1:n , i) * A(i , i+1:n)
```



Pivoting in Gaussian Elimination

- **A = [0 1] fails completely because can't divide by A(1,1)=0**
  [1 0]

- **But solving Ax=b should be easy!**

- **When diagonal A(i,i) is tiny (not just zero), algorithm may terminate but get completely wrong answer**
  - **Numerical instability**
  - **Roundoff error is cause**

- **Cure: Pivot (swap rows of A) so A(i,i) large**

Gaussian Elimination with Partial Pivoting (GEPP)

- Partial Pivoting: swap rows so that A(i,i) is largest in column

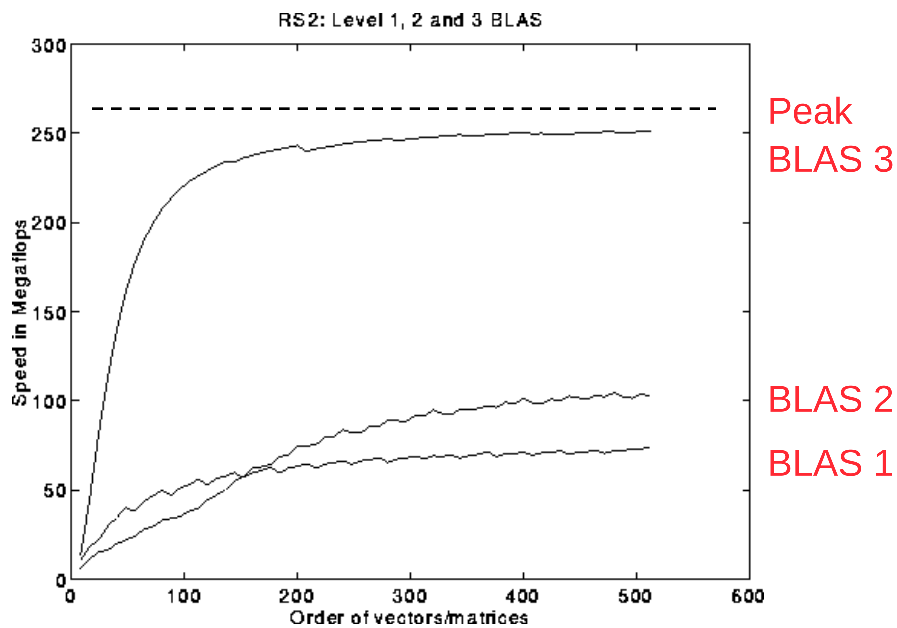
```
for i = 1 to n-1
    find and record k where |A(k,i)| = max{i <= j <= n} |A(j,i)|
          … i.e. largest entry in rest of column i
    if |A(k,i)| = 0
        exit with a warning that A is singular, or nearly so
    elseif  k != i
        swap rows i and k of A
    end if
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)        … each |quotient| ≤ 1
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n ) - A(i+1:n , i) * A(i , i+1:n)
```

- *Lemma*: This algorithm computes A = P*L*U, where P is a permutation matrix.
- This algorithm is numerically stable in practice
- For details see LAPACK code at http://www.netlib.org/lapack/single/sgetf2.f

Problems with basic GE algorithm

- What if some A(i,i) is zero? Or very small?
  - Result may not exist, or be "unstable", so need to pivot
- Current computation all BLAS 1 or BLAS 2, but we know that BLAS 3 (matrix multiply) is fastest (earlier lectures…)

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)          … BLAS 1 (scale a vector)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )   … BLAS 2 (rank-1 update)
            - A(i+1:n , i) * A(i , i+1:n)
```



Converting BLAS2 to BLAS3 in GEPP

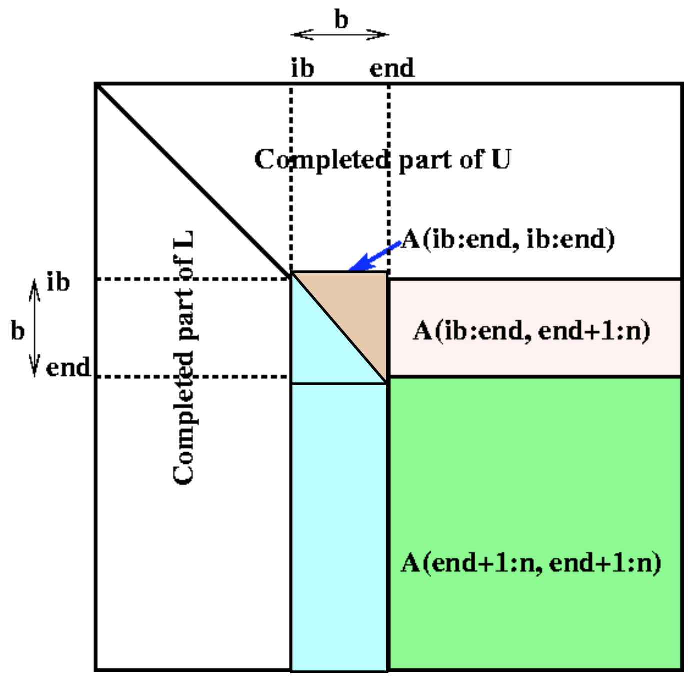
- **Blocking**
  - Used to optimize matrix-multiplication
  - Harder here because of data dependencies in GEPP
- **BIG IDEA: Delayed Updates**
  - Save updates to "trailing matrix" from several consecutive BLAS2 updates
  - Apply many updates simultaneously in one BLAS3 operation
- **Same idea works for much of dense linear algebra**
  - Open questions remain
- **First Approach: Need to choose a block size b**
  - Algorithm will save and apply b updates
  - b must be small enough so that active submatrix consisting of b columns of A fits in cache
  - b must be large enough to make BLAS3 fast

Blocked GEPP (www.netlib.org/lapack/single/sgetrf.f)

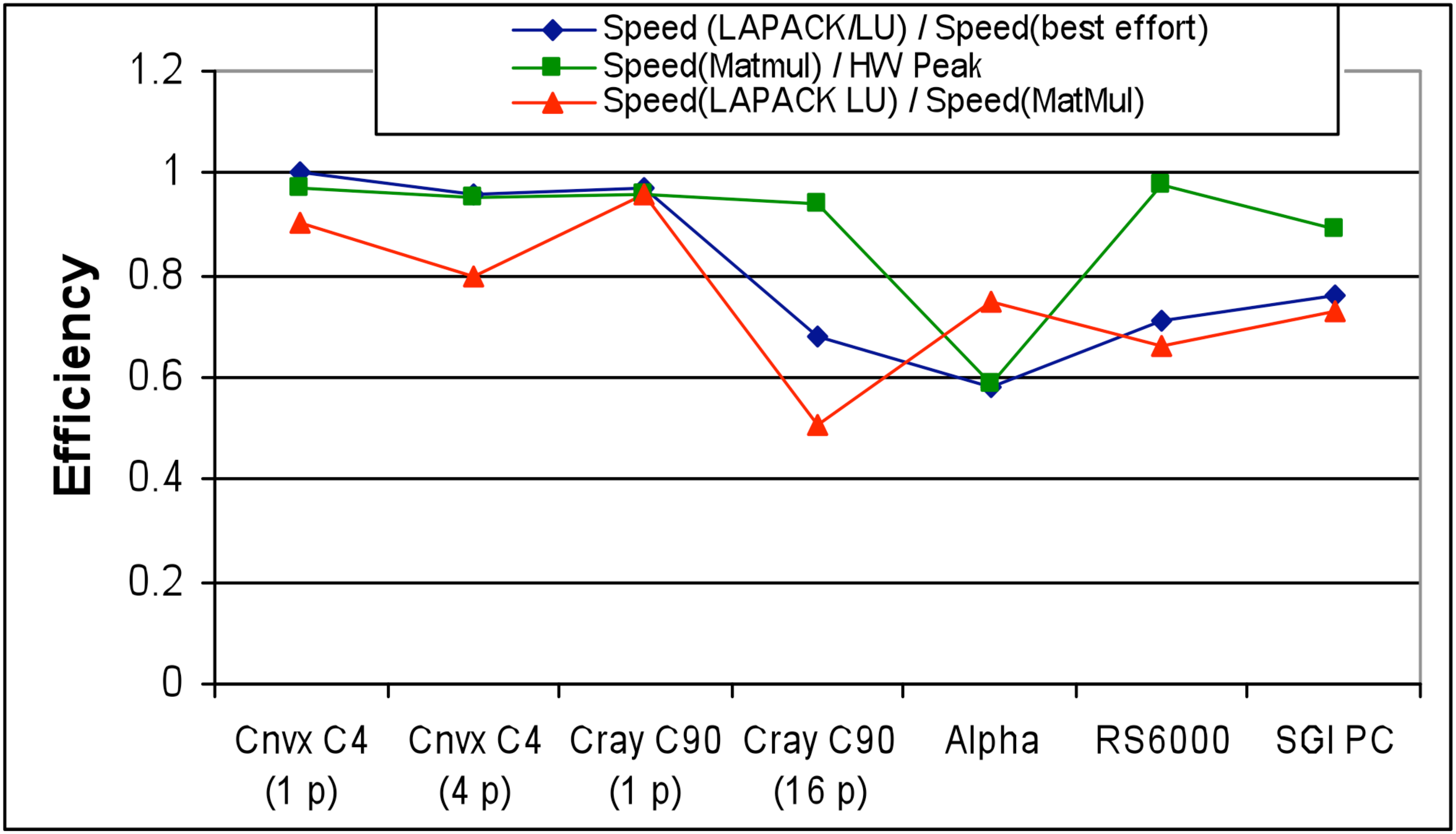
```
for  ib = 1 to n-1 step b     … Process matrix b columns at a time
   end = ib + b-1             … Point to end of block of b columns
   apply BLAS2 version of GEPP to get A(ib:n , ib:end) = P' * L' * U'
   … let LL denote the strict lower triangular part of A(ib:end , ib:end) + I
   A(ib:end , end+1:n) = LL^-1 * A(ib:end , end+1:n)      … update next b rows of U
   A(end+1:n , end+1:n ) = A(end+1:n , end+1:n )
            - A(end+1:n , ib:end) * A(ib:end , end+1:n)
                 … apply delayed updates with single matrix-multiply
                 … with inner dimension b
```

(For a correctness proof, see on-line notes from CS267 / 1996.)

Gaussian Elimination using BLAS 3



Efficiency of Blocked GEPP
(all parallelism "hidden" inside the BLAS)



Outline of rest of talk

- ScaLAPACK GEPP
- Multicore GEPP
- Rest of DLA what's it like (not GEPP)
- Missing from ScaLAPACK - projects
- Design space more generally
- projects

Explicitly Parallelizing Gaussian Elimination

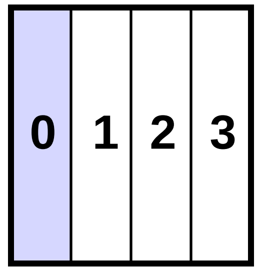
- **Parallelization steps**
  - **Decomposition:** identify enough parallel work, but not too much
  - **Assignment:** load balance work among threads
  - **Orchestrate:** communication and synchronization
  - **Mapping:** which processors execute which threads (locality)
- **Decomposition**
  - In BLAS 2 algorithm nearly each flop in inner loop can be done in parallel, so with n^2 processors, need 3n parallel steps, O(n log n) with pivoting

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)          … BLAS 1 (scale a vector)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )  … BLAS 2 (rank-1 update)
            - A(i+1:n , i) * A(i , i+1:n)
```

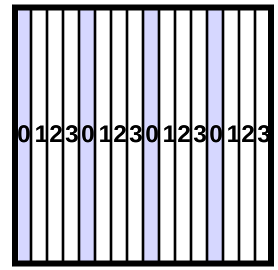
  - This is too fine-grained, prefer calls to local matmuls instead
  - Need to use parallel matrix multiplication
- **Assignment and Mapping**
  - Which processors are responsible for which submatrices?

Different Data Layouts for Parallel GE

Bad load balance:
P0 idle after first
n/4 steps



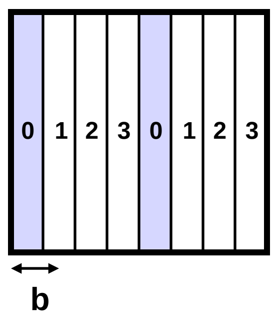
1) 1D Column Blocked Layout



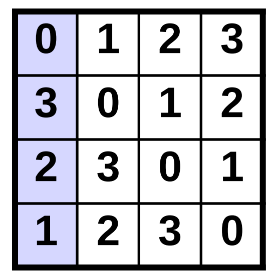
Load balanced, but
can't easily use
BLAS2 or BLAS3

2) 1D Column Cyclic Layout

Can trade load balance
and BLAS2/3
performance by
choosing b, but
factorization of block
column is a bottleneck



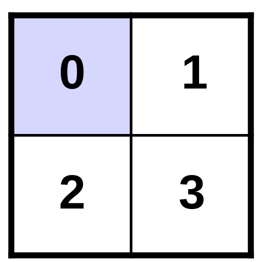
3) 1D Column Block Cyclic Layout



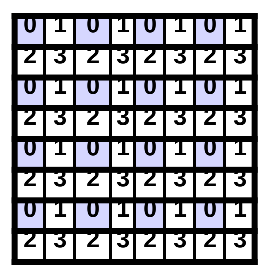
Complicated addressing,
May not want full parallelism
In each column, row

4) Block Skewed Layout

Bad load balance:
P0 idle after first
n/2 steps



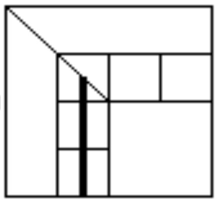
5) 2D Row and Column Blocked Layout

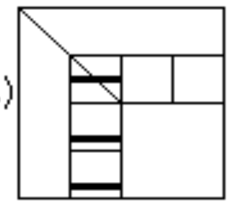


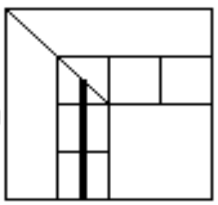
The winner!

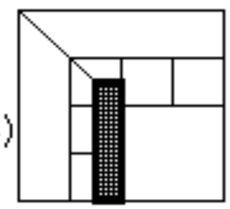
6) 2D Row and Column
Block Cyclic Layout

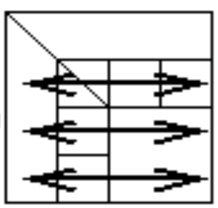
Distributed Gaussian Elimination with a 2D Block Cyclic Layout

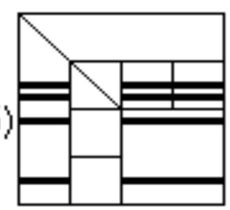
(1) 

(2) 

(3) 

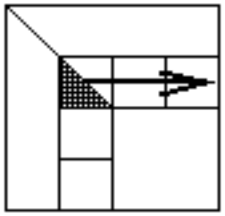
(4) 

(5) 

(6) 

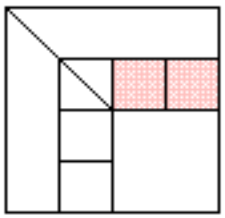
```
for ib = 1 to n-1 step b

    end = min( ib+b-1, n )

    for i = ib to end

    (1)   find pivot row k, column broadcast

    (2)   swap rows k and i in block column, broadcast row k

    (3)   A( i+1:n , i ) = A( i+1:n , i ) / A( i , i )

    (4)   A( i+1:n , i+1:end) -= A( i+1:n , i ) * A( i , i+1:end )

    end for

(5)   broadcast all swap information right and left

(6)   apply all rows swaps to other columns
```

(7)



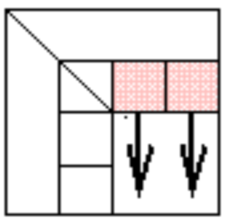
(7) Broadcast LL right

(8)



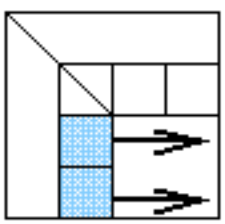
(8) A(ib:end , end+1:n) = LL \ A(ib:end , end+1:n)

(9)



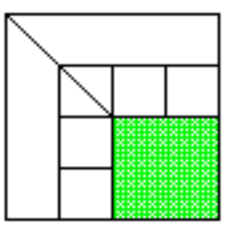
(9) Broadcast A(ib:end , end+1:n) down

(10)



(10) Broadcast A(end+1:n , ib:end) right

(11)



(11) Eliminate A(end+1:n , end+1:n)

Matrix multiply of
green = green - blue * pink

Review of Parallel MatMul

- **Want Large Problem Size Per Processor**

PDGEMM = PBLAS matrix multiply

Observations:

- **For fixed N, as P increasesn Mflops increases, but less than 100% efficiency**
- **For fixed P, as N increases, Mflops (efficiency) rises**

DGEMM = BLAS routine for matrix multiply

Maximum speed for PDGEMM = # Procs * speed of DGEMM

Observations:

- **Efficiency always at least 48%**
- **For fixed N, as P increases, efficiency drops**
- **For fixed P, as N increases, efficiency increases**

Performance of PBLAS

Speed in Mflops of PDGEMM					
Machine	Procs	Block Size	N 2000	N 4000	N 10000
Cray T3E	4=2x2	32	1055	1070	0
	16=4x4		3630	4005	4292
	64=8x8		13456	14287	16755
IBM SP2	4	50	755	0	0
	16		2514	2850	0
	64		6205	8709	10774
Intel XP/S MP Paragon	4	32	330	0	0
	16		1233	1281	0
	64		4496	4864	5257
Berkeley NOW	4	32	463	470	0
	32=4x8		2490	2822	3450
	64		4130	5457	6647

Efficiency = MFlops(PDGEMM)/(Procs*MFlops(DGEMM))						
Machine	Peak/proc	DGEMM Mflops	Procs	N 2000	N 4000	N 10000
Cray T3E	600	360	4	.73	.74	
			16	.63	.70	.75
			64	.58	.62	.73
IBM SP2	266	200	4	.94		
			16	.79	.89	
			64	.48	.68	.84
Intel XP/S MP Paragon	100	90	4	.92		
			16	.86	.89	
			64	.78	.84	.91
Berkeley NOW	334	129	4	.90	.91	
			32	.60	.68	.84
			64	.50	.66	.81

PDGESV = ScaLAPACK Parallel LU

Since it can run no faster than its inner loop (PDGEMM), we measure:
Efficiency = Speed(PDGESV)/Speed(PDGEMM)

Observations:

- Efficiency well above 50% for large enough problems
- For fixed N, as P increases, efficiency decreases (just as for PDGEMM)
- For fixed P, as N increases efficiency increases (just as for PDGEMM)
- From bottom table, cost of solving
  - Ax=b about half of matrix multiply for large enough matrices.
  - From the flop counts we would expect it to be $(2*n^3)/(2/3*n^3) = 3$ times faster, but communication makes it a little slower.

Performance of ScaLAPACK LU

Efficiency = MFlops(PDGESV)/MFlops(PDGEMM)					
Machine	Procs	Block Size	N 2000	N 4000	N 10000
Cray T3E	4	32	.67	.82	
	16		.44	.65	.84
	64		.18	.47	.75
IBM SP2	4	50	.56		
	16		.29	.52	
	64		.15	.32	.66
Intel XP/S MP Paragon	4	32	.64		
	16		.37	.66	
	64		.16	.42	.75
Berkeley NOW	4	32	.76		
	32		.38	.62	.71
	64		.28	.54	.69

Time(PDGESV)/Time(PDGEMM)					
Machine	Procs	Block Size	N 2000	N 4000	N 10000
Cray T3E	4	32	.50	.40	
	16		.75	.51	.40
	64		1.86	.72	.45
IBM SP2	4	50	.60		
	16		1.16	.64	
	64		2.24	1.03	.51
Intel XP/S GP Paragon	4	32	.52		
	16		.89	.50	
	64		2.08	.79	.44
Berkeley NOW	4	32	.44		
	32		.88	.54	.47
	64		1.18	.62	.49

ScaLAPACK Performance Models (1)

ScaLAPACK Operation Counts

$$T(N,P) = \frac{C_f N^3}{P} t_f + \frac{C_v N^2}{\sqrt{P}} t_v + \frac{C_m N}{NB} t_m, \qquad T_{seq}(N,P) = C_f N^3 t_f.$$

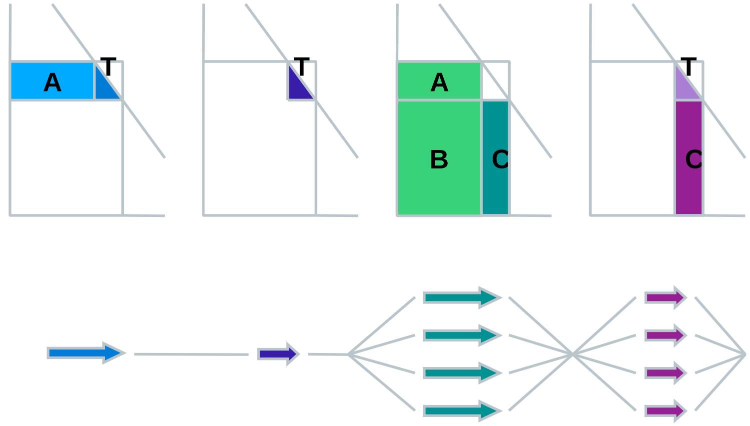
$$E(N,P) = \left(1 + \frac{1}{NB} \frac{C_m t_m}{C_f t_f} \frac{P}{N^2} + \frac{C_v t_v}{C_f t_f} \frac{\sqrt{P}}{N}\right)^{-1}.$$

t_f = 1
t_m = α
t_v = β
NB = brow=bcol
√P = prow = pcol

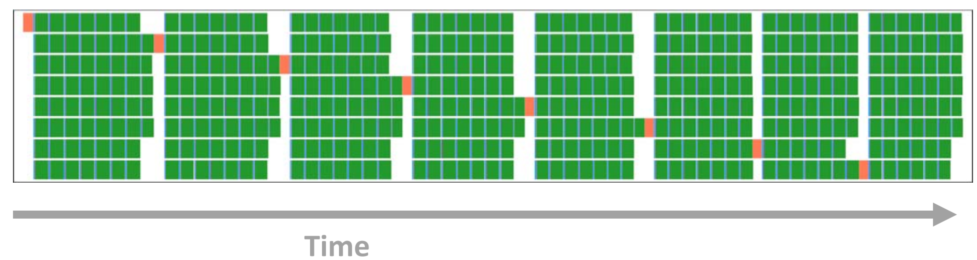
Driver	Options	C_f	C_v	C_m
PxGESV	1 right hand side	$2/3$	$3 + 1/4 \log_2 P$	$NB\,(6 + \log_2 P)$
PxPOSV	1 right hand side	$1/3$	$2 + 1/2 \log_2 P$	$4 + \log_2 P$
PxGELS	1 right hand side	$4/3$	$3 + \log_2 P$	$2\,(NB \log_2 P + 1)$
PxSYEVX	eigenvalues only	$4/3$	$5/2 \log_2 P$	$17/2\, NB + 2$
PxSYEVX	eigenvalues and eigenvectors	$10/3$	$5 \log_2 P$	$17/2\, NB + 2$
PxSYEV	eigenvalues only	$4/3$	$5/2 \log_2 P$	$17/2\, NB + 2$
PxSYEV	eigenvalues and eigenvectors	$22/3$	$5 \log_2 P$	$17/2\, NB + 2$
PxGESVD	singular values only	$26/3$	$10 \log_2 P$	$17 NB$
PxGESVD	singular values and left and right singular vectors	$38/3$	$14 \log_2 P$	$17 NB$
PxLAHQR	eigenvalues only	5	$9/2(\sqrt{P}) * \log_2 P$ $+8\, N/NB$	$9\,(2 + \log_2 P)\, N$
PxLAHQR	full Schur form	18	$9/2(\sqrt{P}) * \log_2 P$ $+8\, N/NB$	$9\,(2 + \log_2 P)\, N$

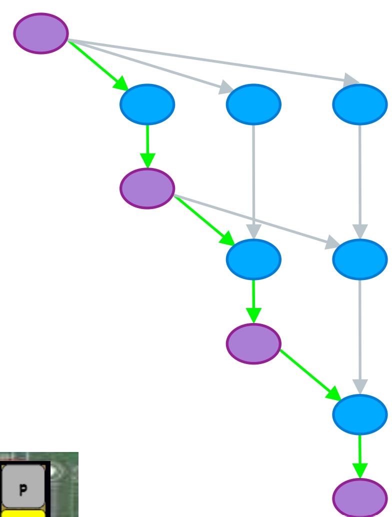
Fork-Join vs. Dynamic Execution

Source: Jack Dongarra

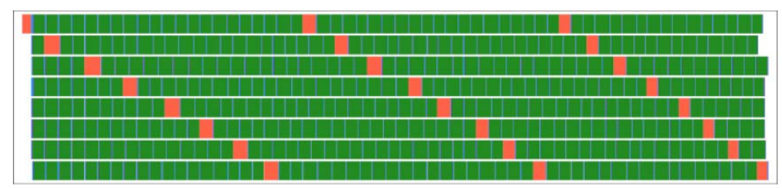


Fork-Join – parallel BLAS

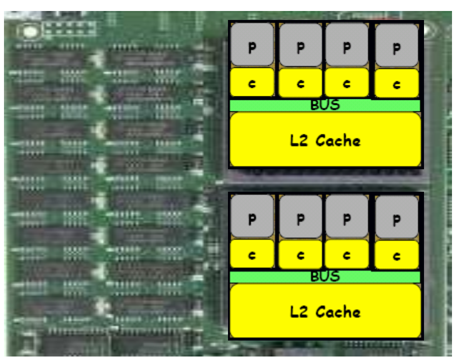




DAG-based – dynamic scheduling



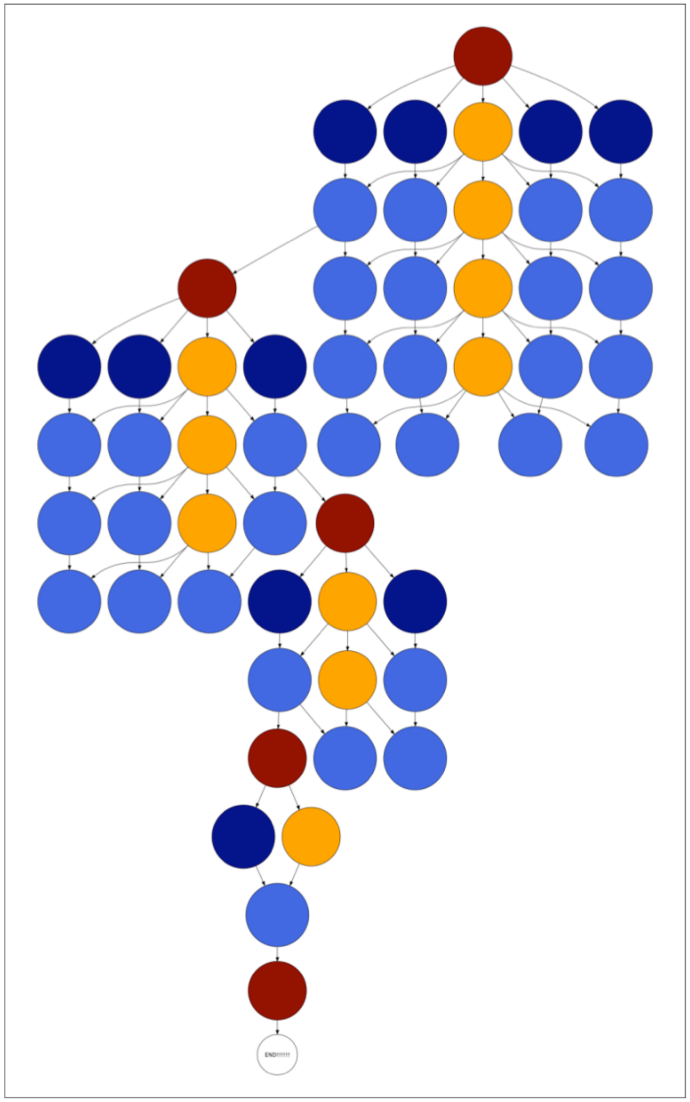
Time saved



Experiments on Intel's Quad Core Clovertown with 2 Sockets w/ 8 Treads

Achieving Asynchronicity

Source: Jack Dongarra



The matrix factorization can be represented as a DAG:

- **nodes:** tasks that operate on "tiles"
- **edges:** dependencies among tasks

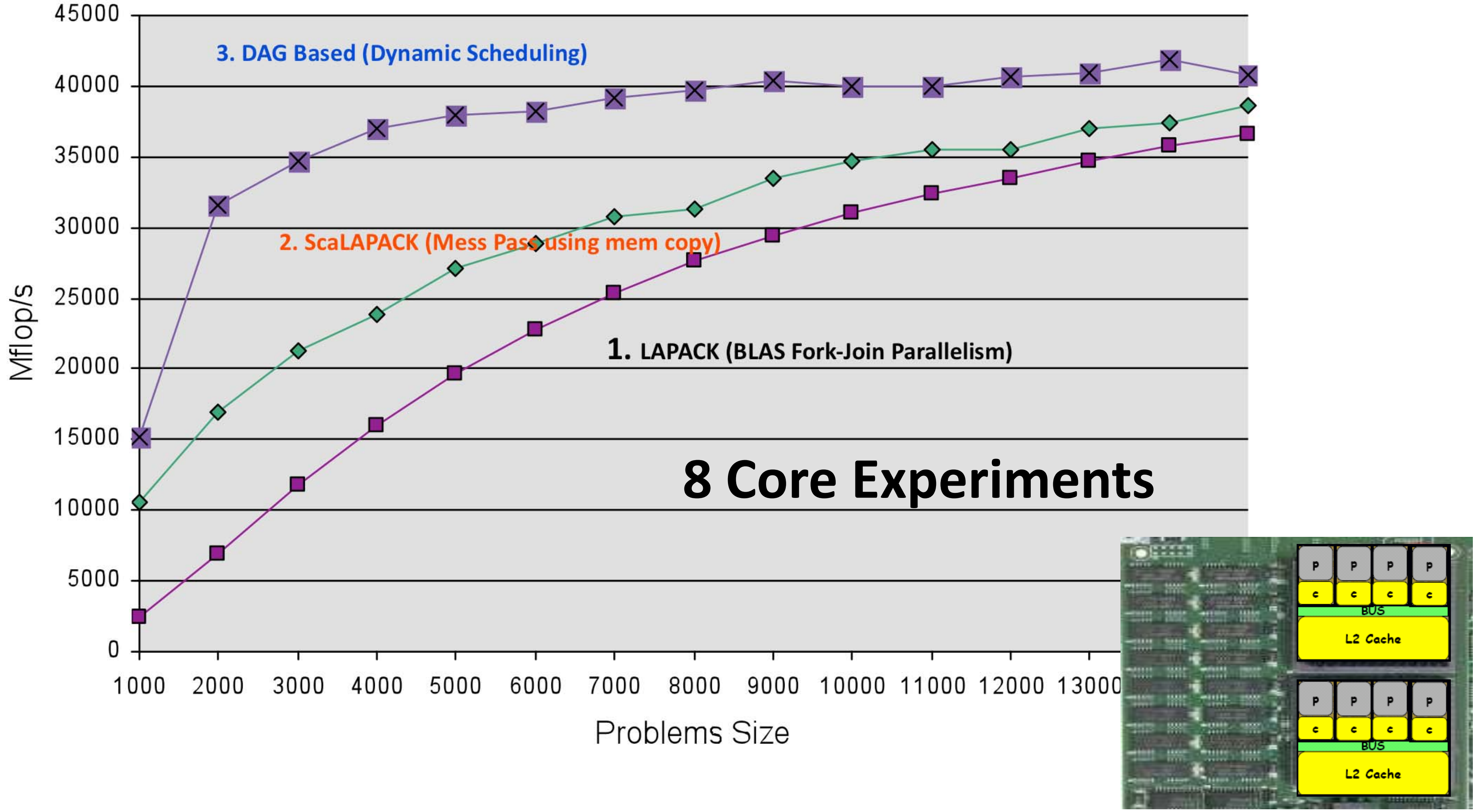
Tasks can be scheduled asynchronously and in any order as long as dependencies are not violated.

System: PLASMA

Intel's Clovertown Quad Core

Source: Jack Dongarra

3 Implementations of LU factorization
Quad core w/2 sockets per board, w/ 8 Treads



LAPACK and ScaLAPACK Scalability

- **"One-sided Problems" are scalable**
  - **Linear systems Ax=b, and least squares $\min_x ||Ax-b||_2$**
  - **In Gaussian elimination, A factored into product of 2 matrices A = LU by premultiplying A by sequence of simpler matrices**
  - **Asymptotically 100% BLAS3**
  - **LU ("Linpack Benchmark")**
  - **Cholesky, QR**
- **"Two-sided Problems" are harder**
  - **Eigenvalue problems, SVD**
  - **A factored into product of 3 matrices by pre and post multiplication**
  - **~Half BLAS2, not all BLAS3**
- **Narrow band problems hardest (to do BLAS3 or parallelize)**
  - **Solving and eigenvalue problems**

What *could* go into a linear algebra library?

For all linear algebra problems

For all matrix/problem structures

For all data types

For all architectures and networks

For all programming interfaces

Produce best algorithm(s) w.r.t.
performance and accuracy
(including condition estimates, etc)

Need to prioritize, automate!

Missing Routines in Sca/LAPACK

		LAPACK	ScaLAPACK
Linear Equations	LU	xGESV	PxGESV
	LU + iterative refine	xGESVX	missing
	Cholesky	xPOSV	PxPOSV
	LDL^T	xSYSV	missing
Least Squares (LS)	QR	xGELS	PxGELS
	QR+pivot	xGELSY	missing
	SVD/QR	xGELSS	missing
	SVD/D&C	xGELSD	missing (intent?)
	SVD/MRRR	missing	missing
	QR + iterative refine.	missing	missing
Generalized LS	LS + equality constr.	xGGLSE	missing
	Generalized LM	xGGGLM	missing
	Above + Iterative ref.	missing	missing

More missing routines

		LAPACK	ScaLAPACK
Symmetric EVD	QR / Bisection+Invit D&C MRRR	xSYEV / X xSYEVD xSYEVR	PxSYEV / X PxSYEVD missing
Nonsymmetric EVD	Schur form Vectors too	xGEES / X xGEEV /X	missing (driver) missing
SVD	QR D&C MRRR Jacobi	xGESVD xGESDD missing missing	PxGESVD missing (intent?) missing missing
Generalized Symmetric EVD	QR / Bisection+Invit D&C MRRR	xSYGV / X xSYGVD missing	PxSYGV / X missing (intent?) missing
Generalized Nonsymmetric EVD	Schur form Vectors too	xGGES / X xGGEV / X	missing missing
Generalized SVD	Kogbetliantz MRRR	xGGSVD missing	missing (intent) missing

Exploring the tuning space for Dense LA

- **Algorithm tuning space includes**
  - **Underlying BLAS (PHiPAC, ATLAS)**
  - **Different layouts (blocked, recursive, …) and algorithms**
  - **Numerous block sizes, not just in underlying BLAS**
  - **Many possible layers of parallelism, many mappings to HW**
  - **Different traversals of underlying DAGs**
    - **Synchronous and asynchronous algorithms**
  - **“Redundant” algorithms for GPUs**
  - **New and old eigenvalue algorithms**
  - **Mixed precision (for speed or accuracy)**
  - **New “communication avoiding” algorithms for variations on standard factorizations**
- **Is there a concise set of abstractions to describe, generate tuning space?**
  - **Block matrices, factorizations (partial, tree, …), DAGs, …**
  - **PLASMA, FLAME, CSS, Spiral, Sequoia, Telescoping languages, Bernoulli, Rose, …**
- **Question: What fraction of dense linear algebra can be generated/tuned?**
  - **Lots more than when we started**
    - **Sequential BLAS -> Parallel BLAS -> LU -> other factorizations -> …**
  - **Most of dense linear algebra?**
    - **Not eigenvalue algorithms (on compact forms)**
    - **What fraction of LAPACK can be done?**
    - **“for all linear algebra problems…”**
  - **For all interesting architectures…?**

Possible class projects

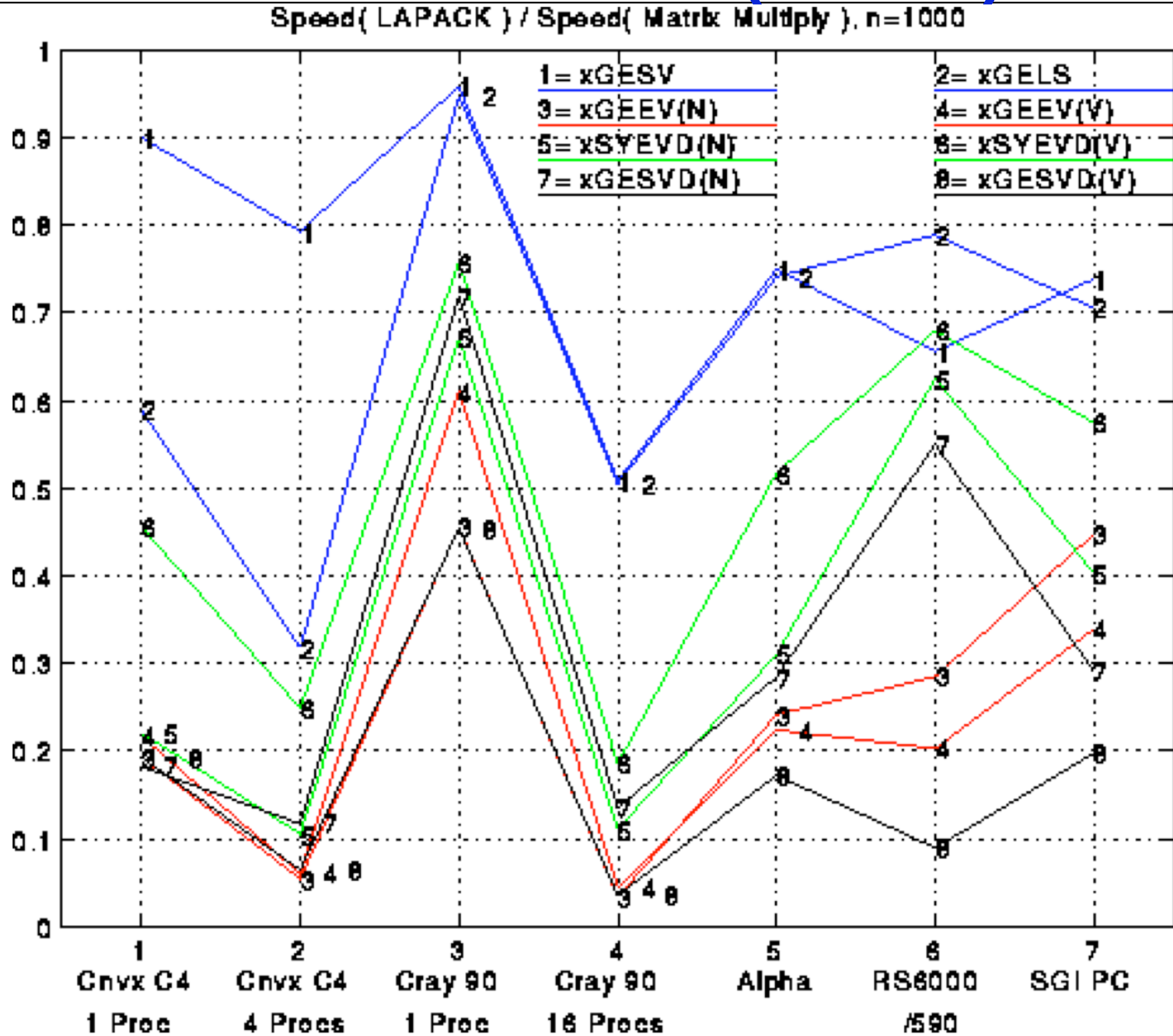
- **GPU related**
  - **Best results so far do some work on GPU, some on CPU**
  - **Try porting algorithms to NVIDIA GPU using CUDA**
  - **Explore mixed precision algorithms**
- **Filling in gaps in ScaLAPACK**
  - **User demand for various missing routines**
- **Eigenvalues routines on Multicore**
  - **Compare performance of LAPACK, ScaLAPACK**
  - **Explore multithreaded implementations (PLASMA?)**
- **New "communication avoiding" QR algorithm**
  - **Implement, compare performance to Sca/LAPACK**
  - **Try in eigenvalues routines**
  - **Try analogous LU routine**
- **Study code automation systems**
  - **List on previous slide**
- **More at**
  - **www.cs.berkeley.edu/~demmel/Sca-LAPACK-Proposal.pdf**

Extra Slides

Overview of LAPACK and ScaLAPACK

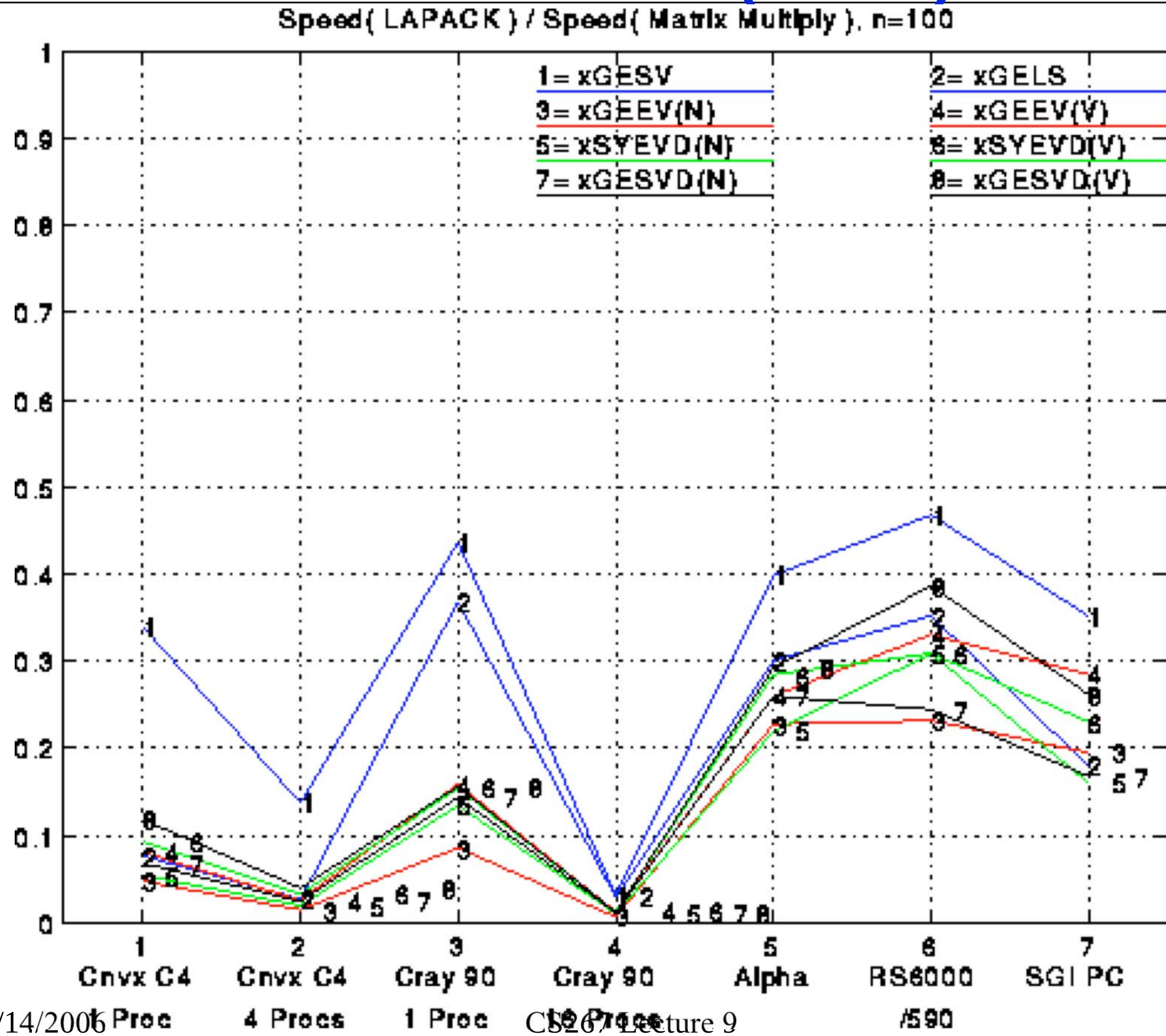
- **Standard library for dense/banded linear algebra**
  - **Linear systems: $A*x=b$**
  - **Least squares problems: $\min_x \| A*x-b \|_2$**
  - **Eigenvalue problems: $Ax = \lambda x$, $Ax = \lambda Bx$**
  - **Singular value decomposition (SVD): $A = U\Sigma V^T$**
- **Algorithms reorganized to use BLAS3 as much as possible**
- **Basis of math libraries on many computers, Matlab …**
- **Many algorithmic innovations remain**
  - **Projects available**

Performance of LAPACK (n=1000)



Performance of Eigen-values, SVD, etc.

Performance of LAPACK (n=100)

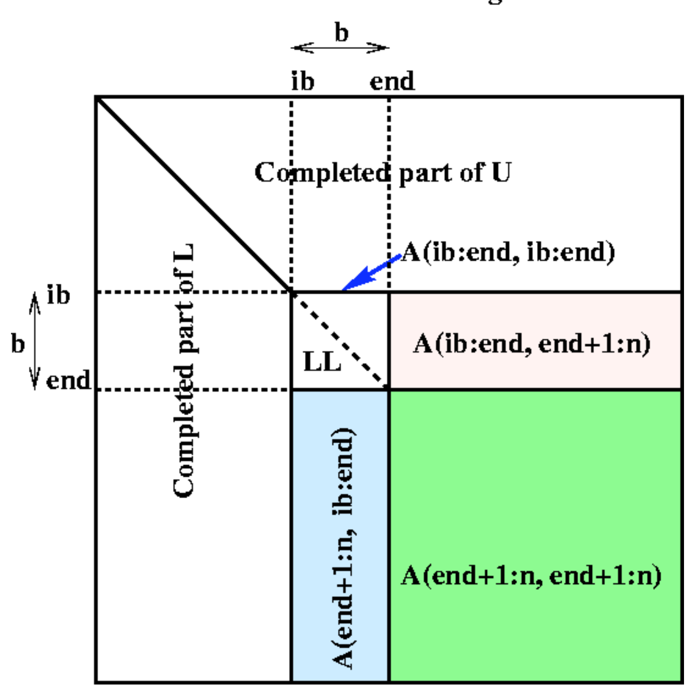


Efficiency is much lower for a smaller matrix.

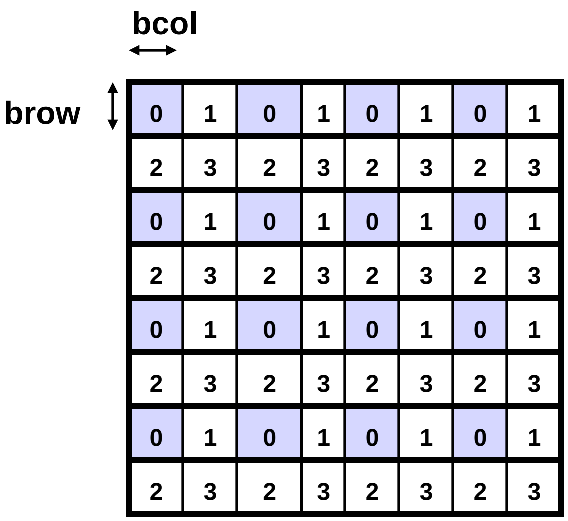
Review: BLAS 3 (Blocked) GEPP

```
for   ib = 1 to n-1 step b      … Process matrix b columns at a time
    end = ib + b-1              … Point to end of block of b columns
    apply BLAS2 version of GEPP to  get A(ib:n , ib:end) = P' * L' * U'
    … let LL denote the strict lower triangular part of A(ib:end , ib:end) + I
          A(ib:end , end+1:n) = LL^-1 * A(ib:end , end+1:n)        … update next b rows of U
BLAS 3 {  A(end+1:n , end+1:n ) = A(end+1:n , end+1:n )
                    - A(end+1:n , ib:end) * A(ib:end , end+1:n)
                                   … apply delayed updates with single matrix-multiply
                                   … with inner dimension b
```

Gaussian Elimination using BLAS 3



Row and Column Block Cyclic Layout



- **processors and matrix blocks are distributed in a 2d array**
  - **prow-by-pcol array of processors**
  - **brow-by-bcol matrix blocks**
- **pcol-fold parallelism in any column, and calls to the BLAS2 and BLAS3 on matrices of size brow-by-bcol**
- **serial bottleneck is eased**
- **prow ≠ pcol and brow ≠ bcol possible, even desireable**

Distributed GE with a 2D Block Cyclic Layout

- **block size b in the algorithm and the block sizes brow and bcol in the layout satisfy b=bcol.**
- **shaded regions indicate processors busy with computation or communication.**
- **unnecessary to have a barrier between each step of the algorithm, e.g.. steps 9, 10, and 11 can be pipelined**

ScaLAPACK Performance Models (2)

Compare Predictions and Measurements

IBM SP2[a]	P	Values of N									
		2000		5000		7500		10000		15000	
		Est	Obt	Est	Obt	Est	Obt	Est	Obt	Est	Obt
PDGESV	4	357	421	632	603						
(LU)	16	497	722	1581	1543	2116	1903	2424	2149		
	64	502	924	2432	3017	4235	4295	5793	5596	7992	7057
PDPOSV	4	530	462	669	615						
(Cholesky)	16	1315	1081	2083	1811	2366	2118	2535	2312		
	64	2577	1807	5327	4431	6709	5727	7661	6826	8887	8084

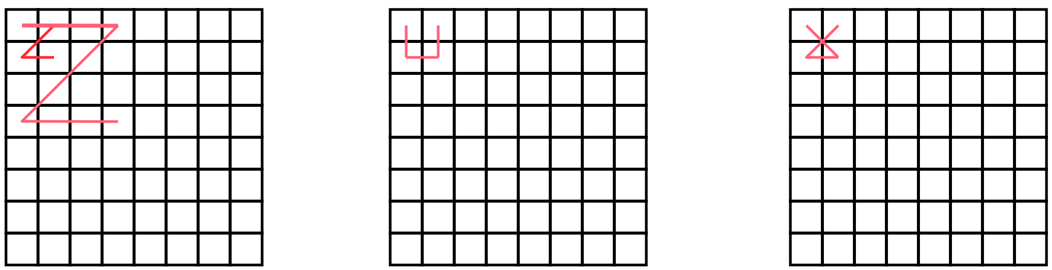
[a] One process spawned per node and one computational IBM POWER2 590 processor per node.

Next release of LAPACK and ScaLAPACK

- **Class projects available**
- **www.cs.berkeley.edu/~demmel/Sca-LAPACK-Proposal.pdf**
- **New or improved LAPACK algorithms**
  - **Faster and/or more accurate routines for linear systems, least squares, eigenvalues, SVD**
- **Parallelizing algorithms for ScaLAPACK**
  - **Many LAPACK routines not parallelized yet**
- **Automatic performance tuning**
  - **Many tuning parameters in code**

Recursive Algorithms

- **Still uses delayed updates, but organized differently**
  - **(formulas on board)**
- **Can exploit recursive data layouts**
  - **3x speedups on least squares for tall, thin matrices**



- **Theoretically optimal memory hierarchy performance**
- **See references at**
  - **"Recursive Block Algorithms and Hybrid Data Structures," Elmroth, Gustavson, Jonsson, Kagstrom, SIAM Review, 2004**
  - **http://www.cs.umu.se/research/parallel/recursion/**

Gaussian Elimination via a Recursive Algorithm

F. Gustavson and S. Toledo

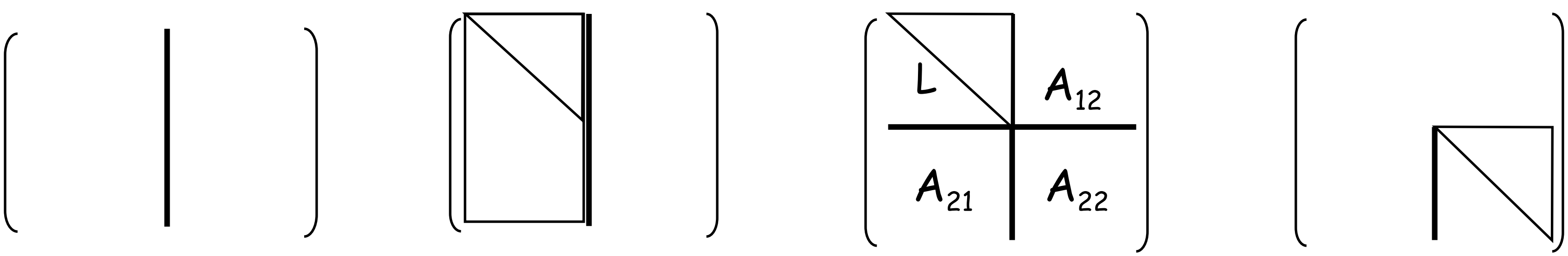
LU Algorithm:

1: Split matrix into two rectangles (m x n/2)
   if only 1 column, scale by reciprocal of pivot & return

2: Apply LU Algorithm to the left part

3: Apply transformations to right part
   (triangular solve $A_{12} = L^{-1}A_{12}$ and
   matrix multiplication $A_{22}=A_{22} - A_{21} * A_{12}$)

4: Apply LU Algorithm to right part

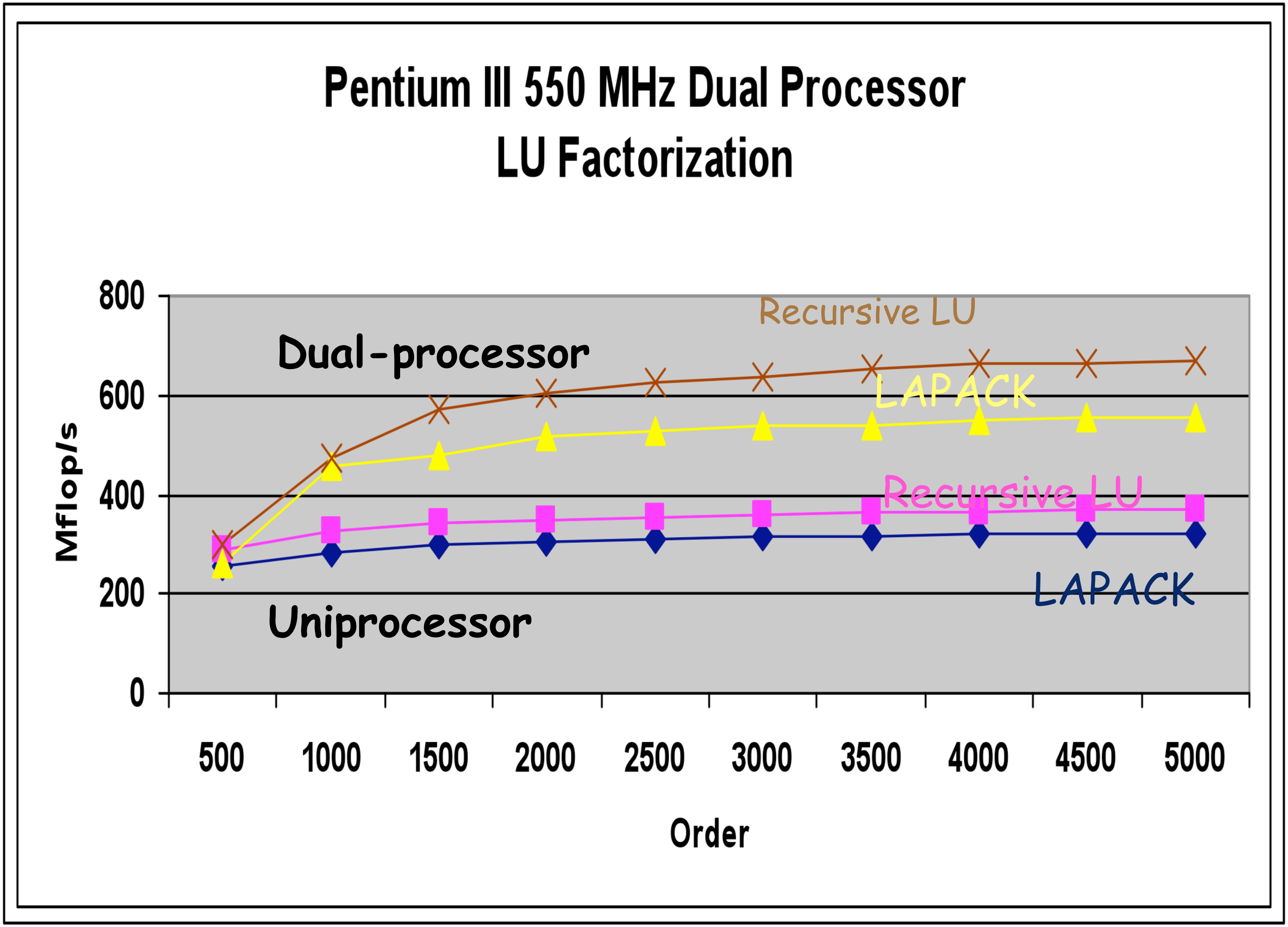


Most of the work in the matrix multiply
Matrices of size n/2, n/4, n/8, …

Source: Jack Dongarra

Recursive Factorizations

- Just as accurate as conventional method
- Same number of operations
- Automatic variable-size blocking
  - Level 1 and 3 BLAS only !
- Simplicity of expression
- Potential for efficiency while being “cache oblivious”
  - But shouldn’t recur down to single columns!
- The recursive formulation is just a rearrangement of the point -wise LINPACK algorithm
- The standard error analysis applies (assuming the matrix operations are computed the “conventional” way).



Source: Jack Dongarra

Recursive Algorithms – Limits

- **Two kinds of dense matrix compositions**
- **One Sided**
  - **Sequence of simple operations applied on left of matrix**
  - **Gaussian Elimination: A = L*U or A = P*L*U**
    - **Symmetric Gaussian Elimination: $A = L*D*L^T$**
    - **Cholesky: $A = L*L^T$**
  - **QR Decomposition for Least Squares: A = Q*R**
  - **Can be nearly 100% BLAS 3**
  - **Susceptible to recursive algorithms**
- **Two Sided**
  - **Sequence of simple operations applied on both sides, alternating**
  - **Eigenvalue algorithms, SVD**
  - **At least ~25% BLAS 2**
  - **Seem impervious to recursive approach?**
  - **Some recent progress on SVD (25% vs 50% BLAS2)**

Out of "Core" Algorithms

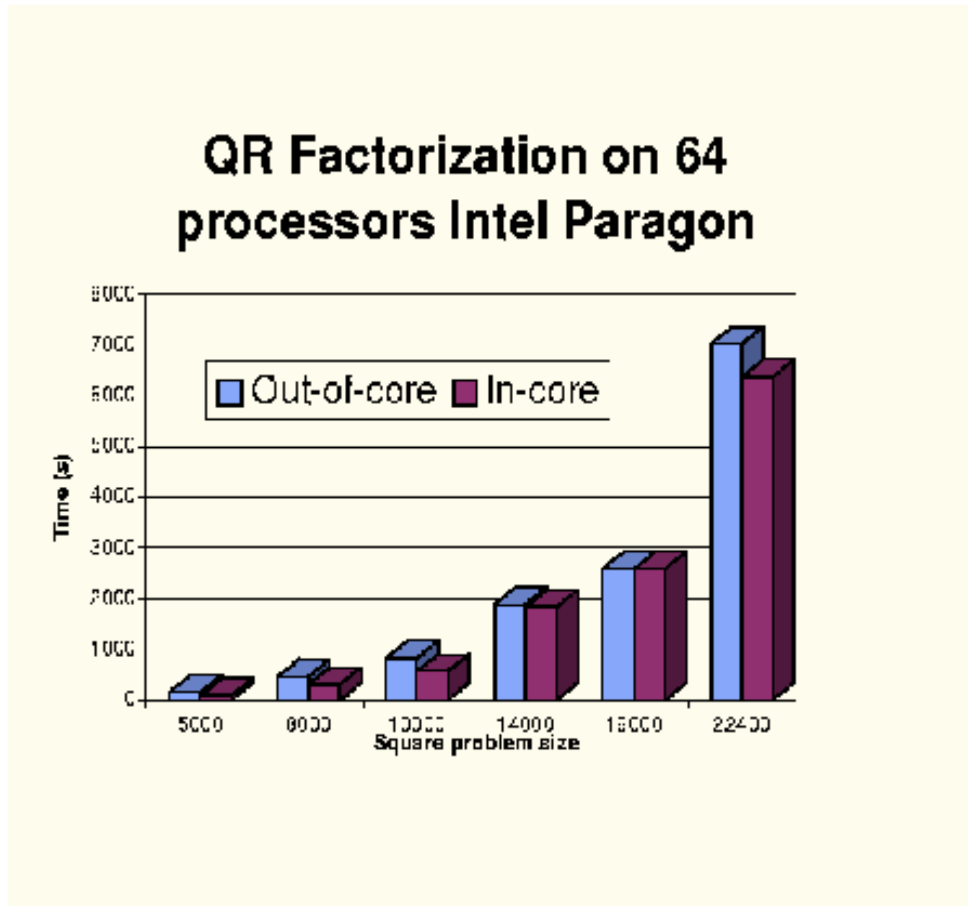
Out-of-core means matrix lives on disk; too big for main memory

Much harder to hide latency of disk

QR much easier than LU because no pivoting needed for QR

Out-of-Core Performance Results for Least Squares

- **Prototype code for Out-of-Core extension**
- **Linear solvers based on "Left-looking" variants of LU, QR, and Cholesky factorization**
- **Portable I/O interface for reading/writing ScaLAPACK matrices**



Source: Jack Dongarra

Some contributors (incomplete list)

Participants

Krste Asanovic (UC Berkeley)
Richard Barrett (U. Tenn)
Jeff Bilmes (UC Berkeley)
Susan Blackford (ORNL)
Tony Chan (UCLA)
Jaeyoung Choi (LBNL)
Ed D'Azevedo (ORNL)
Inderjit Dhillon (UC Berkeley)
Jack Dongarra (U Tenn, ORNL)
Jeremy Du Croz (NAG)
Stan Eisenstat (Yale)
John Gilbert (Xerox PARC)
Sven Hammarling (NAG)
Greg Henry (Intel)
Steve Huss-Lederman (SRC)
W. Kahan (UC Berkeley)
Rencang Li (UC Berkeley)
Joseph Liu (York)
Antoine Petitet (U Tenn)
Roldan Pozo (U Tenn)
Huan Ren (UC Berkeley)
Charles Romine (ORNL)
Ivan Slapničar (U Split)
Ken Stanley (UC Berkeley)
Bernard Tourancheau (U Tenn)
Robert van de Geijn (U Texas)
Paul Van Dooren (U Illinois)
David Walker (ORNL)
Kathy Yelick (UC Berkeley)
Zhaojun Bai (U Kentucky)
Michael Berry (U Tenn)
Chris Bischof (ANL)
Soumen Chakrabarti (UC Berkeley)
Chee-Whye Chin (UC Berkeley)
Andy Cleary (LLNL)
Jim Demmel (UC Berkeley)
June Donato (ORNL)
Zlatko Drmač (U Hagen)
Victor Eijkhout (UCLA)
Vince Fernando (NAG)
Ming Gu (UC Berkeley, LBL)
Mike Heath (U Illinois)
Dominic Lam (UC Berkeley)
Bo Kågström (U Umeå)
Youngbae Kim (U Tenn)
Xiaoye Li (UC Berkeley)
Beresford Parlett (UC Berkeley)
Peter Poromaa (U Umeå)
Padma Raghavan (U Illinois)
Howard Robinson (UC Berkeley)
Jeff Rutter (UC Berkeley)
Dan Sorensen (Rice U)
Xiaobai Sun (ANL)
Anna Tsao (SRC)
Henk van der Vorst (Utrecht U)
Krešimir Veselić (U Hagen)
Clint Whaley (U Tenn)

With the cooperation of
Cray, IBM, Convex, DEC, Fujitsu, NEC, NAG, IMSL

Supported by ARPA, NSF, DOE

Upcoming related talks

- **SIAM Conference on Parallel Processing in Scientific Computing**
  - **San Francisco, Feb 22-24**
  - **http://www.siam.org/meetings/pp06/index.htm**
  - **Applications, Algorithms, Software, Hardware**
  - **3 Minisymposia on Dense Linear Algebra on Friday 2/24**
    - **MS41, MS47(*), MS56**
- **Scientific Computing Seminar,**
  - **"An O(n log n) tridiagonal eigensolver", Jonathan Moussa**
  - **Wednesday, Feb 15, 11-12, 380 Soda**
- **Special Seminar**
  - **Towards Combinatorial Preconditioners for Finite -Elements Problems", Prof. Sivan Toledo, Technion**
  - **Tuesday, Feb 21, 1-2pm, 373 Soda**

Extra Slides

QR (Least Squares)

Performance of ScaLAPACK QR (Least squares)

Scales well,
nearly full machine speed

Efficiency = MFlops(PDGELS)/MFlops(PDGEMM)					
Machine	Procs	Block Size	N		
			2000	4000	10000
Cray T3E	4	32	.54	.61	
	16		.46	.55	.60
	64		.26	.47	.54
IBM SP2	4	50	.51		
	16		.29	.51	
	64		.19	.36	.54
Intel XP/S GP Paragon	4	32	.61		
	16		.43	.63	
	64		.22	.48	.62
Berkeley NOW	4	32	.51	.77	
	32		.49	.66	.71
	64		.37	.60	.72

Time(PDGELS)/Time(PDGEMM)					
Machine	Procs	Block Size	N		
			2000	4000	10000
Cray T3E	4	32	1.2	1.1	
	16		1.5	1.2	1.1
	64		2.6	1.4	1.2
IBM SP2	4	50	1.3		
	16		2.3	1.3	
	64		3.6	1.8	1.2
Intel XP/S GP Paragon	4	32	1.1		
	16		1.6	1.1	
	64		3.0	1.4	1.1
Berkeley NOW	4	32	1.3	.9	
	32		1.4	1.0	.9
	64		1.8	1.1	.9

Performance of Symmetric Eigensolvers

Current algorithm:
Faster than initial algorithm
Occasional numerical instability
New, faster and more stable algorithm planned

Time(PDSYEVX)/Time(PDGEMM) (bisection + inverse iteration)				
Machine	Procs	Block Size	N 2000	N 4000
Cray T3E	4	32	10	
	16		13	10
	64		29	14
IBB SP2	16	50	24	
	64		40	29
Intel XP/S GP Paragon	16	32	22	
	64		34	20
Berkeley NOW	16	32	20	
	32		24	52

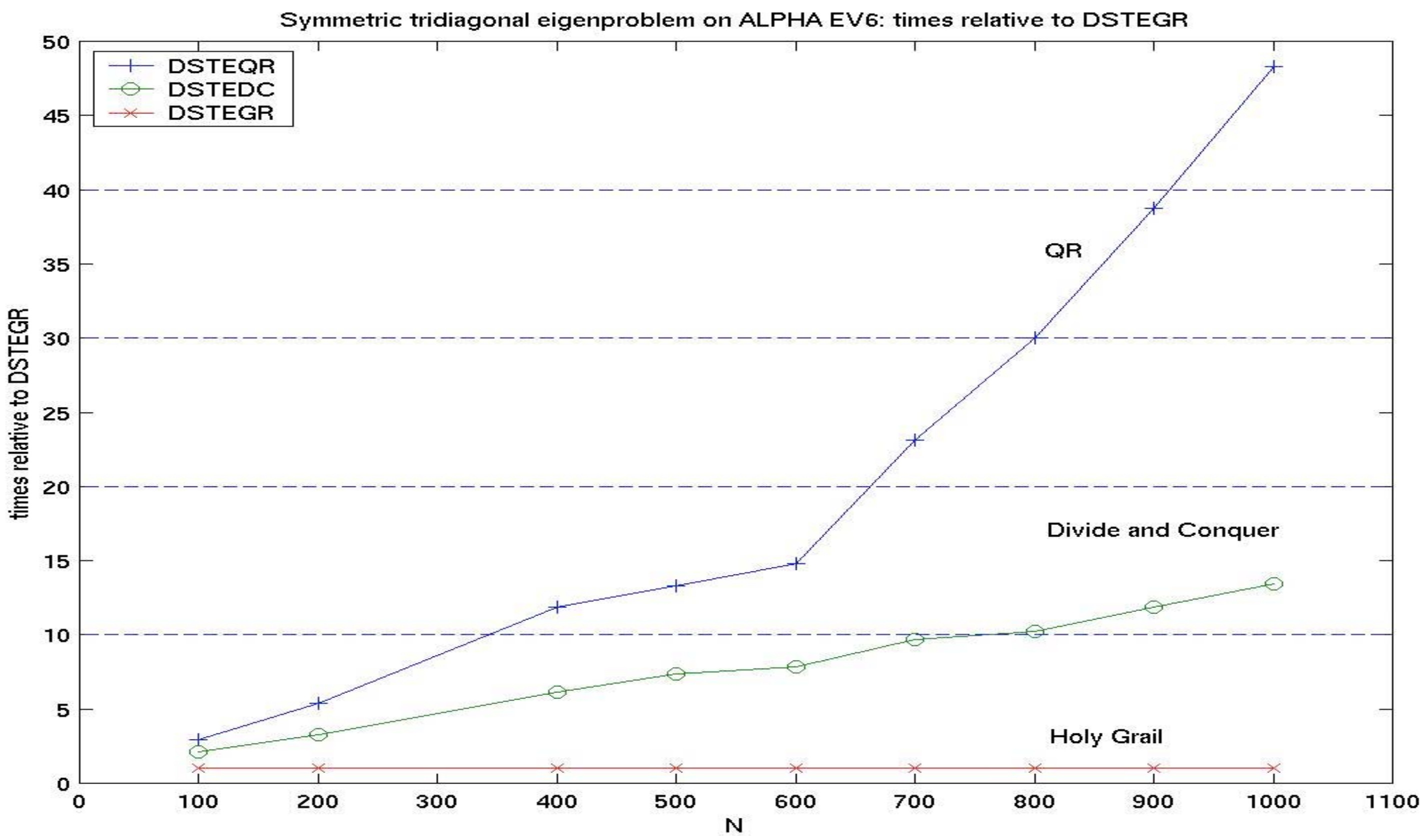
Initial algorithm:
Numerically stable
Easily parallelized
Slow: will abandon

Time(PDSYEV)/Time(PDGEMM) (QR iteration)				
Machine	Procs	Block Size	N 2000	N 4000
Cray T3E	4	32	35	
	16		37	35
	64		57	41
IBM SP2	16	50	38	
	64		58	47
Intel XP/S GP Paragon	16	32	99	
	64		193	
Berkeley NOW	16	32	31	
	32		35	55

Scalable Symmetric Eigensolver and SVD

The "Holy Grail" (Parlett, Dhillon, Marques)

Perfect Output complexity (O(n * #vectors)), Embarrassingly parallel, Accurate



To be propagated throughout LAPACK and ScaLAPACK

Performance of SVD
(Singular Value Decomposition)

Have good ideas to speedup
Project available!

Time(PDGESVD)/Time(PDGEMM)				
Machine	Procs	Block Size	N 2000	N 4000
Cray T3E	4	32	67	
	16		66	64
	64		93	70
IBM SP2	4	50	97	
	16		60	
	64		81	
Berkeley NOW	4	32	72	
	16		38	16
	32		59	26

Performance of Nonsymmetric Eigensolver
(QR iteration)

Hardest of all to parallelize

Time(PDLAHQR)/Time(PDGEMM)				
Machine	Procs	Block Size	N 1000	N 1500
Intel XP/S MP Paragon	16	50	123	97

Scalable Nonsymmetric Eigensolver

- $Ax_i = \lambda_i x_i$, Schur form $A = QTQ^T$
- Parallel HQR
  - Henry, Watkins, Dongarra, Van de Geijn
  - Now in ScaLAPACK
  - Not as scalable as LU: N times as many messages
  - Block-Hankel data layout better in theory, but not in ScaLAPACK
- Sign Function
  - Beavers, Denman, Lin, Zmijewski, Bai, Demmel, Gu, Godunov, Bulgakov, Malyshev
  - $A_{i+1} = (A_i + A_i^{-1})/2 \rightarrow$ shifted projector onto Re $\lambda > 0$
  - Repeat on transformed A to divide-and-conquer spectrum
  - Only uses inversion, so scalable
  - Inverse free version exists (uses QRD)
  - Very high flop count compared to HQR, less stable

Assignment of parallel work in GE

- Think of assigning submatrices to threads, where each thread responsible for updating submatrix it owns
  - "owner computes" rule natural because of locality
- What should submatrices look like to achieve load balance?

Computational Electromagnetics (MOM)

The main steps in the solution process are

- **Fill: computing the matrix elements of A**
- **Factor: factoring the dense matrix A**
- **Solve: solving for one or more excitations b**
- **Field Calc: computing the fields scattered from the object**

Analysis of MOM for Parallel Implementation

Task	Work	Parallelism	Parallel Speed
Fill	O(n**2)	embarrassing	low
Factor	O(n**3)	moderately diff.	very high
Solve	O(n**2)	moderately diff.	high
Field Calc.	O(n)	embarrassing	high

BLAS2 version of GE with Partial Pivoting (GEPP)

```
for i = 1 to n-1
    find and record k where |A(k,i)| = max{i <= j <= n} |A(j,i)|
        … i.e. largest entry in rest of column i
    if |A(k,i)| = 0
        exit with a warning that A is singular, or nearly so
    elseif  k != i
        swap rows i and k of A
    end if
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)
                … each quotient lies in [-1,1]
                … BLAS 1
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n ) - A(i+1:n , i) * A(i , i+1:n)
                … BLAS 2, most work in this line
```

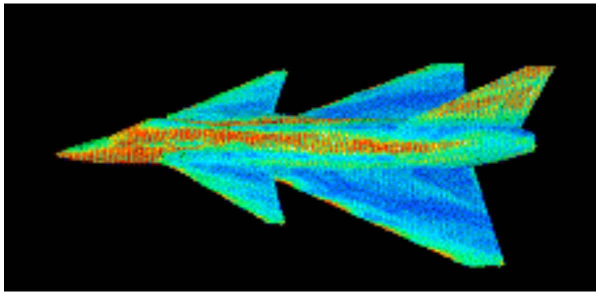
Computational Electromagnetics – Solve Ax=b

- Developed during 1980s, driven by defense applications
- Determine the RCS (radar cross section) of airplane
- Reduce signature of plane (stealth technology)
- Other applications are antenna design, medical equipment
- Two fundamental numerical approaches:
    - MOM methods of moments (frequency domain)
        - Large dense matrices
    - Finite differences (time domain)
        - Even larger sparse matrices

Computational Electromagnetics

- Discretize surface into triangular facets using standard modeling tools

- Amplitude of currents on surface are unknowns



- Integral equation is discretized into a set of linear equations

image: NW Univ. Comp. Electromagnetics Laboratory http://nueml.ece.nwu.edu/

Computational Electromagnetics (MOM)

After discretization the integral equation has the form

$$\mathbf{A\ x\ =\ b}$$

where

A is the (dense) impedance matrix,

x is the unknown vector of amplitudes, and

b is the excitation vector.

(see Cwik, Patterson, and Scott, Electromagnetic Scattering on the Intel Touchstone Delta, IEEE Supercomputing ‘92, pp 538 - 542)

Results for Parallel Implementation on Intel Delta

Task	Time (hours)
Fill (compute n^2 matrix entries)	9.20
(embarrassingly parallel but slow)	
Factor (Gaussian Elimination, $O(n^3)$)	8.25
(good parallelism with right algorithm)	
Solve ($O(n^2)$)	2 .17
(reasonable parallelism with right algorithm)	
Field Calc. ($O(n)$)	0.12
(embarrassingly parallel and fast)	

The problem solved was for a matrix of size 48,672.
2.6 Gflops for Factor - The world record in 1991.

Computational Chemistry – $Ax = \lambda x$

- **Seek energy levels of a molecule, crystal, etc.**
  - **Solve Schroedinger's Equation for energy levels = eigenvalues**
  - **Discretize to get $Ax = \lambda Bx$, solve for eigenvalues λ and eigenvectors x**
  - **A and B large Hermitian matrices (B positive definite)**
- **MP-Quest (Sandia NL)**
  - **Si and sapphire crystals of up to 3072 atoms**
  - **A and B up to n=40000, complex Hermitian**
  - **Need all eigenvalues and eigenvectors**
  - **Need to iterate up to 20 times (for self-consistency)**
- **Implemented on Intel ASCI Red**
  - **9200 Pentium Pro 200 processors (4600 Duals, a CLUMP)**
  - **Overall application ran at 605 Gflops (out of 1800 Gflops peak),**
  - **Eigensolver ran at 684 Gflops**
  - **www.cs.berkeley.edu/~stanley/gbell/index.html**
  - **Runner-up for Gordon Bell Prize at Supercomputing 98**

LAPACK and ScaLAPACK

	LAPACK	ScaLAPACK
Machines	Workstations, Vector, SMP	Distributed Memory, DSM
Based on	BLAS	BLAS, BLACS
Functionality	Linear Systems Least Squares Eigenproblems	Linear Systems Least Squares Eigenproblems (less than LAPACK)
Matrix types	Dense, band	Dense, band, out-of-core
Error Bounds	Complete	A few
Languages	F77 or C	F77 and C
Interfaces to	C++, F90	HPF
Manual?	Yes	Yes
Where?	www.netlib.org/ lapack	www.netlib.org/ scalapack

Parallelism in ScaLAPACK

- Level 3 BLAS block operations
  - All the reduction routines
- Pipelining
  - QR Iteration, Triangular Solvers, classic factorizations
- Redundant computations
  - Condition estimators
- Static work assignment
  - Bisection
- Task parallelism
  - Sign function eigenvalue computations
- Divide and Conquer
  - Tridiagonal and band solvers, symmetric eigenvalue problem and Sign function
- Cyclic reduction
  - Reduced system in the band solver

Winner of TOPS 500 (LINPACK Benchmark)

Year	Machine	Tflops	Factor faster	Peak Tflops	Num Procs	N
2004	Blue Gene / L, IBM	70.7	2.0	91.8	32768	.93M
200 220 03	Earth System Computer, NEC	35.6	4.9	40.8	5104	1.04M
2001	ASCI White, IBM SP Power 3	7.2	1.5	11.1	7424	.52M
2000	ASCI White, IBM SP Power 3	4.9	2.1	11.1	7424	.43M
1999	ASCI Red, Intel PII Xeon	2.4	1.1	3.2	9632	.36M
1998	ASCI Blue, IBM SP 604E	2.1	1.6	3.9	5808	.43M
1997	ASCI Red, Intel Ppro, 200 MHz	1.3	3.6	1.8	9152	.24M
1996	Hitachi CP-PACS	.37	1.3	.6	2048	.10M
1995	Intel Paragon XP/S MP	.28	1	.3	6768	.13M

Source: Jack Dongarra (UTK)